Graphes d'intervalles : Correction

I Introduction

I.A Représentation du problème

Question I.A.1.

```
let conflit (a,b) (c,d) =
    c <= b && a <= d
;;
```

I.B Graphe simple non orienté

R.A.S.

I.C Graphe d'intervalles

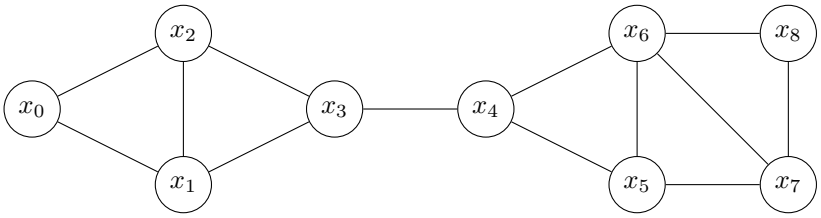

Figure 3

Question I.C.1.

Question I.C.2.

```
let construit_graphe t =
    let n = Array.length t in
    let out = Array.make n [] in
    for i = 0 to (n-2) do
        for j = (i+1) to (n-1) do
            if conflit t.(i) t.(j) then
                out.(i) <- j::out.(i) ;
                out.(j) <- i::out.(j)
        done
    done;
    out
;;
```

I.D Coloration

Question I.D.1. *Pour le problème a la suite* $(0,1,2,0,1,0,0)$ *convient.*
Pour le problème b la suite $(0,1,2,0,1,0,2,1,0)$ *convient.*

Question I.D.2. ***Couleur disponible***

a)
```
let rec appartient l x =
      match l with
       |[] -> false
       |a::tl -> if a=x then true else appartient tl x
;;
```

Remarque. Cette fonction n'est non pas de signature **int list -> int -> bool** mais de signature **'a list -> 'a -> bool**.... mais en vrai on s'en balance.

b)
```
let plus_petit_abscent l =
      let n = ref 0 in
      while appartient l !n do
          n := !n+1
      done;
      !n
;;
```

Remarque. Une version récursive est possible afin d'éviter l'utilisation de référence.

c)
```
let couleurs_voisins arete couleurs i =
       let rec boucle l1 l2 =
           match l1 with
            |[] -> l2
            |a::tl -> if couleurs.(a) = -1 then boucle tl l2 else boucle tl couleurs.(a)::l2
       in
       boucle arete.(i) []
   ;;
```
d)
```
let couleur_disponible aretes couleurs i =
       plus_petit_abscent (couleurs_voisins aretes couleurs i)
   ;;
```

I.E Cliques

Question I.E.1.

a) $\chi(G) = 1$ *et* $\omega(G) = 1$

b) $\chi(G) = n$ *et* $\omega(G) = n$

Question I.E.2. *On a clairement* $\chi(G) \geqslant \omega(G)$*, en effet dès que l'on a affaire à une clique de taille* c *il nous faut nécessairement au moins* c *couleurs différentes pour colorier le graphe.*

L'inverse n'est pas vrai. Prenons cinq sommet connectés en un grand cycle, ainsi la plus grande clique possible est de taille 2 alors qu'il nous faudra 3 couleurs pour colorier cela.

Question I.E.3.
```
let rec est_clique aretes xs =
    let rec boucle sommet liste =
       match liste with
        |[] -> true
        |a::tl -> appartient (aretes.(a)) sommet && boucle sommet tl
    in
    match xs with
     |[] -> true
     |a::tl -> boucle a tl && est_clique aretes tl
;;
```

II Algorithme glouton pour la coloration

II.A L'algorithme sur un exemple

Question II.A.1. *Spoiler alert : J'ai utilisé cet algorithme pour répondre à la question* ***I.D.1***

II.B Coloration

Question II.B.1.
```
let coloration segments aretes =
    let n = Array.length segments in
    let out = Array.make n (-1) in
    for i = 0 to (n-1) do
        out.(i) <- couleur_disponible aretes out i
    done;
    out
;;
```

II.C Preuve de l'algorithme

Question II.C.1. *Au moins* c*, en effet, de part le fonctionnement de l'algorithme, le fait que* I_k *reçoive la couleur* c *implique que pour toute couleur comprise entre* 0 *et* $c-1$ *(ensemble vide si* $c = 0$*) notre segment est en intersection avec un segment dans* $\{I_0, \ldots, I_{k-1}\}$ *(ensemble lui aussi vide si* $k = 0$*) de cette couleur. D'où le résultat.*

Question II.C.2. *Les segments étant triés par ordre croissant de leur extrémité gauche, le seul moyen pour qu'il y ait intersection avec un segment précédent est que l'extrémité gauche de* I_k *soit dans le segment en question. Donc tous les voisins de* I_k *d'indice inférieur à* k *contienne la borne gauche de* I_k*. ils ont*

donc tous un élément en commun et donc sont en conflit, nous assurant qu'ils soient tous voisins les uns des autres. Ils forment donc une clique et on a le résultat sur sa taille d'après la question précédente.

Question II.C.3. *D'après la question précédente, on vient de trouver une clique de taille $c+1$. D'où le résultat d'après la question **I.E.2**.*

Question II.C.4. *Si on reformule le résultat de la question précédente on obtient : à chaque étape de l'algorithme on ajoute une couleur c telle que $c+1$ soit inférieur ou égal au nombre minimum de couleurs $\chi(G)$ requises pour colorier le graphe. D'où, à la dernière étape le résultat suivant : toute couleur utilisée par l'algorithme appartient à $[|0, \chi(G)|]$, d'où le résultat.*

II.D Complexité

Question II.D.1. *Pour chaque sommet du graphe on appel **couleur_disponible** cette dernière va elle-même appeler **couleurs_voisins** qui va regarder les couleurs de chacun des voisins de ce sommet. On pourrait optimiser en suivant à la lettre l'algorithme décrit en début de section afin de parcourir ici une seule fois chaque arête, mais assymptotiquement cela ne change pas grand chose, on va juste regarder deux fois chaque arête au lieu d'une seule. On sera donc en $O(m)$ dans tous les cas.*

*Attention, il y a aussi l'appel à **plus_petit_absent**, mais le nombre de couleur minimum est clairement borné par deux fois le nombre d'arêtes (plus un, dans le cas ou il y a zéro arête).*

D'où une complexité en $O(m)$.

III Graphes munis d'un ordre d'élimination parfait

III.A Un exemple

Question III.A.1. *L'énumération $(x_4, x_5, x_2, x_1, x_6, x_3, x_7, x_0)$ convient.*

III.B Vérification

Question III.B.1.

```
let voisins_inferieurs aretes x =
    let rec boucle l1 l2 =
        match l1 with
         |[] -> l2
         |a::tl -> if a<x then boucle tl (a::l2) else boucle tl l2
    in
    boucle aretes.(x) []
;;
```

Question III.B.2.

```
let est_ordre_parfait aretes =
    let out = ref true in
    for x = 0 to (Array.length aretes - 1) do
        if not (est_clique aretes (voisins_inferieurs aretes x)) then
            out := false
    done;
    !out
;;
```

III.C Ordre d'élimination parfait pour un graphe d'intervalles

Question III.C.1. *c.f question **II.C.2***

III.D Coloration

Question III.D.1.

a) $(0, 1, 0, 1, 2, 3, 2, 0)$

*b) avec l'énumération de la question **III.A.1*** $(1, 3, 2, 1, 0, 1, 0, 2)$

Question III.D.2.

```
let rec liste_couleur l tab =
    match l with
     |[] -> []
     |a::tl -> tab.(a)::(liste_couleur tl tab)
;;

let colore aretes =
    let n = Array.length aretes in
    let c = Array.make n (-1) in
    for i = 0 to (n-1) do
        c.(i) <- plus_petit_abscent (liste_couleur (voisins_inferieurs aretes i) c)
    done;
    c
;;
```

Remarque. Cette question est un peu embêtante car elle fat complètement doublon avec la **II.B.1**. Ici je me suis amusé à trouver une solution faisant appel à la fonction **voisins_inferieurs** mais en fait vous pourriez redonner exactement le même algorithme (avec l'argument "segments" en moins), puisque finalement les deux algorithmes demandent juste de colorier les sommets dans leur ordre d'apparitions dans le tableau "aretes".

Question III.D.3.

a) *Puisqu'on a affaire à un ordre d'élimination parfait le sommet i et ses voisins d'indice plus petit forment une clique de taille au plus $\omega(G)$ donc la couleur c_i prise par le sommet i sera au plus $\omega(G)-1$, puisqu'il y a alors forcément au moins une couleur de libre parmi $[|0,\omega(G)-1|]$, donc $1+c_i \leqslant \omega(G)$. Or, d'après la question **I.E.2**, on a $\chi(G) \geqslant \omega(G)$.*
Q.E.D.

b) *D'après la question précédente toutes les couleurs sont comprises entre 0 et $\chi(G)-1$, il y en a donc au mieux $\chi(G)$ d'où le résultat.*

IV Ordre d'élimination parfait pour un graphe cordal

IV.A Cycles de longueur 4 dans un graphe d'intervalles

Question IV.A.1. *Traitons le cas de I_0, le résultat sera valide pour les autres par symétrie des rôles de chacun des segments.*

Par hypothèse I_0 ne peut être inclus dans I_2. Ensuite, si I_0 est inclus dans I_1, comme $I_0 \cap I_3 \neq \emptyset$ la borne gauche ou la borne droite de I_3 est dans I_0 et donc serait dans I_1 ce qui contredit l'hypothèse $I_1 \cap I_3 = \emptyset$. De même, par un argument similaire mais sur les bornes de I_1, supposer que I_0 est inclus dans I_3 contredit la même hypothèse.

Q.E.D.

Question IV.A.2. *Les segments I_1 et I_2 n'ont pas le droit d'être inclus l'un dans l'autre mais leur intersection est non vide. Par conséquent on a soit le résultat soit $min\ I_2 < min\ I_1 < max\ I_2 < max\ I_1$.*

Mais si on se trouve dans le second cas et que $min\ I_0 < min\ I_1 < max\ I_0 < max\ I_1$, on aurait $min\ I_1$ à la fois dans I_0 et I_2 ce qui est une contradiction.

Une fois le résultat pour I_1 et I_2 acquis on applique le même raisonnement pour avoir le résultat sur I_2 et I_3.

Question IV.A.3. *En mettant bout à bout les inégalités de la question précédente on arrive à $max < max\ I_0 < min\ I_3$ et donc $I_0 \cap I_3 = \emptyset$ ce qui est une contradiction.*

IV.B Cordalité des graphes d'intervalles

Question IV.B.1. *Itérons le résultat de la partie précédente.*

C'est à dire : montrons que, pour un $n \geqslant 4$ donné, dans tout graphe d'intervalle tout cycle de longueur n admet une corde.

On le résultat pour $n = 4$

On se donne $n \geqslant 4$ et on suppose le résultat obtenue pour les cycles de longueurs inférieur ou égale à n.

Soit $I_0, \ldots, I_n$ un cycle de longueur $n+1$ dans un graphe d'intervalle. Supposons que ce cycle admette une corde et,quitte à permuter l'ordre des intervalles, on suppose aussi que cette corde n'implique pas les intervalles I_{n-1} et I_n.

On considère alors le graphe d'intervalle formé de l'intervalle $I_{n-1} \cup I_n$ et des autres intervalles de l'ancien graphe. On aura alors un cycle $I_0, \ldots, I_{n-2}, (I_{n-1} \cup I_n)$, dans un nouveau graphe d'intervalle, de longueur n et qui admet toujours l'ancienne corde. Ceci contredit l'hypothèse de récurrence, d'où le résultat.

IV.C Une enquête policière

Question IV.C.1. *Si on dessine le graphe d'intervalles associé à la situation en supposant que personne ne mente on trouve "assez vite" trois cycles de longueur 4 sans corde (ce qui, d'après la partie précédente, pose problème).*

— *Albert, Édouard, Charlotte, Didier*
— *Albert, Didier, Isabelle, Édouard*
— *Albert, Bernard, Isabelle, Didier*

Donc le coupable est parmi l'intersection de ces trois ensemble, c'est à dire Didier ou Albert. le problème c'est qu'on ne peut trouver de cycle sans corde sans l'un de ces deux protagoniste.

nous allons donc nous servir du fait que les quatre autres disent la vérité.

En effet, on peut alors être certain que Bernard et Charlotte ne se sont pas vu et que Isabelle et Édouard sont passés entre ces deux derniers puisqu'ils ont pu voir chacun des trois autres protagonistes innocents.

Sauf que Charlotte a vu Didier et Bernard a vu Albert donc le seul moyen que Didier ait vu Albert soit que ces deux derniers se soient croisés soit quand Isabelle était la soit quand Édouard était la.

Or il n'y a ni cycle Albert, Didier, Édouard, ni cycle Albert, Didier, Isabelle.

Donc Didier et Albert n'était pas présent au même moment. Donc Didier a menti et voilà notre coupable.

C'est bon, vous avez tout suivi ?

IV.D Ordre d'élimination parfait

Question IV.D.1.

```
let simplicial (aretes, sg) k =
    let rec aux liste =
        match liste with
         |[] -> []
         |a::tl -> if sg.(a) then a::(aux tl) else aux tl
    in
    est_clique aretes (aux aretes.(k))
;;
```

*on a commencé par parcourir la liste des voisins de k afin de récupérer la sous liste de ceux étant dans le sous graphe puis on a fait appel la fonction **est_clique** qui elle même va, pour chacun des voisins de k dans le sous-graphe, vérifier si il est bien voisins des autres. Donc on va appeler quadratiquement en le nombre n de sommet la fonction **appartient**.*

On a donc au final une complexité en $O(n^2)$.

Question IV.D.2.

```
let trouver_simplicial (aretes, sg) =
    let sommet = ref -1 in
    for i = 0 to (Array.length sg - 1) do
        if sg.(i) then
            if simplicial (aretes, sg) i then
                sommet := i
    done;
    if !sommet = -1 then failwith "pas de sommet simplicial" else i
;;
```

*On parcours l'ensemble des sommet et on appel **simplicial** pour chacun d'entre eux. On se retrouve donc avec une complexité en $O(n^3)$.*

Question IV.D.3.

```
let ordre_parfait aretes =
    let n = Array.length aretes in
    let sg = Array.make n true in
    let out = ref [] in
    for j = 0 to (n-1) do
```

```
        let i = trouver_simplicial (aretes, sg) in
        sg.(i) <- false;
        out := i::!out
    done;
    !out
;;
```

On appel ici ***trouver_simplicial*** *autant de fois qu'il y a de sommet donc on est en $O(n^4)$.*

Ceci dit, la question n'est pas fini. En effet, avant les algorithme était soit donné soit naturel mais la il n'est pas clair qu'une solution gloutonne comme celle proposée marche. Il va donc nous falloir justifier cet algorithme.

Sa terminaison étant évidente, concentrons nous sur sa validité. Nous allons montrer que si un graphe $G = (S, A)$ admet un ordre d'élimination parfait alors pour tout sommet simplicial x dans G le graphe induit par $S \setminus \{x\}$ admet un ordre d'élimination parfait.

On considère donc le graphe G admettant l'ordre d'élimination parfait $(x_0 \dots, x_{n-1})$ et un sommet simplicial x_i. Si $i = n-1$ le résultat est acquis sinon il suffit de remarquer que $(x_0, \dots, x_{i-1}, x_{i+1}, \dots, x_{n+1})$ est un ordre d'élimination parfait pour le graphe induit.

Q.E.D.

IV.E Coupures minimales dans un graphe cordal

Question IV.E.1. *Prenons le graphe :*

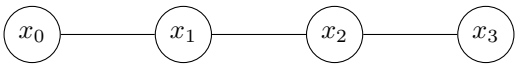

Figure 3

Il est clairement cordal puisqu'il n'a pas de cycle, et l'on peut faire une coupure de cardinal minimale en prenant juste les deux sommets du centre ce qui contre dit le résultat.

Il y a donc une légère erreur d'énoncé. Il faut autoriser les coupures de cardinal 1 afin d'éliminer le contre exemple et que le résultat fasse sens.

Maintenant, considérons dans notre coupure C l'ensemble des sommets C_1 sans voisins dans G_1, alors $C \setminus C_1$ va aussi déconnecter les sommets a et b et donc sera une coupure. Ainsi la minimalité de C oblige $C_1 = \emptyset$ et par symétrie des rôles de x, y, a et b on a le résultat.

Question IV.E.2. *G_1 étant une composante connexe du sous-graphe H tous ses sommets sont reliés les uns au autres. Ainsi si l'on prend a_1 un voisin dans G_1 de x et a_p un voisin de y avec le chemin reliant ces deux sommets dans G_1 on a notre résultat. De même pour le chemin reliant x et y dans G_2.*

Question IV.E.3. *On peut alors considérer le cycle $(x, a_1, \dots, a_p, y, b_1, \dots, b_q, x)$ de longueur au moins 4. Notre graphe étant cordal il existe une corde dans ce cycle.*

Cette corde ne peut relier un sommet intérieur de P_1 avec un de P_2 sinon C ne serait pas une coupe. De plus, si cette corde ne relie pas x et y cela voudrait dire qu'elle relierait deux sommets intérieur de P_1 ou deux sommets intérieur de P_2 et donc qu'en l'empruntant on pourrait crée un cycle répondant aux critères de la question mais plus court contredisant le caractère de minimalité demandé.

D'où le résultat.

Question IV.E.4. *Les questions précédentes montre que si on prend deux sommets de C alors ils sont reliés dans G. Q.E.D.*

IV.F Sommets simpliciaux dans un graphe cordal

Question IV.F.1. *Si G est complet tout sommet est relié à tout autre. Donc si on prend un sommet quelconque, l'ensemble de ses voisins est formé de tous les autres sommets et ils sont tous reliés entre eux deux à deux donc forme une clique.*

Q.E.D.

Question IV.F.2. *Si G n'a qu'un sommet il n'y a rien à montrer, le graphe n'a pas d'arête et les voisins du dit sommet forme $\emptyset$ qui est bien une clique.*

Si G a deux sommet, peut importe le sommet choisit, ses voisins seront uniquement constitués soit de l'autre sommet soit de $\emptyset$ dans les deux cas nous avons affaire à des cliques.

Enfin, si G a trois sommets. Si G est complet le résultat est déjà acquis. Sinon, il y a forcément deux sommets non reliés entre eux. Chacun de ses deux sommets est simplicial. En effet, prenons l'un d'eux, ou alors il n'est relié à rien, auquel cas le résultat est évident, ou alors il est relié au troisième sommet qui, étant tout seul dans l'ensemble de voisin, forme une clique.

Question IV.F.3. *a) Un sous-graphe d'un graphe cordal est clairement cordal car tout cycle existant dans le sous-graphe existe dans le graphe et toute corde de ce cycle dans le graphe existe aussi dans le sous-graphe.*

b) H_1 étant complet tout ses sommets sont simpliciaux, en particulier ceux choisit dans S_1.
De plus, si un tel sommet ne peut avoir de voisin dans S_2 sinon C ne serait pas une coupure. D'où le résultat.

*c) Il s'agit ici simplement d'invoquer la propriété d'induction sur H_1 qui est cordal et a strictement moins de sommet que G, pour avoir la première partie du résultat. De plus, ces deux sommets ne peuvent se trouver dans la coupure en même temps puisque cette dernière est une clique cf question **IV.E.4**. De plus, le sommet dans S_1 ne peut avoir de voisin dans S_2 et par le même raisonnement qu'à la question précédente il est bien simplicial dans G.*

d) Les questions précédentes ont initialisées puis prouver l'hérédité (on a bien DEUX sommets simpliciaux non voisins si le graphe n'est pas complet, un dans H_1 et un dans H_2 par symétrie des rôles de ces derniers) d'un raisonnement par induction (récurrence) d'où le résultat.

IV.G Ordre d'élimination parfait dans un graphe cordal

Question IV.G.1. *On vient de montrer qu'il existe un sommet simplicial dans tout graphe cordal. De plus tout sous-graphe d'un graphe cordal est cordal. il ne nous reste plus qu'à invoquer la preuve de l'algorithme de la question **IV.D.3** pour conclure.*